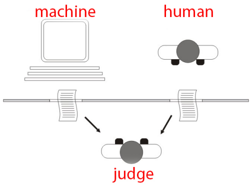
02180: Introduction to Artificial Intelligence

Lecture 6: Monte-Carlo Tree Search

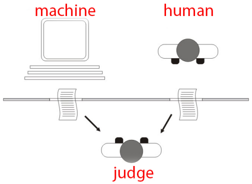
Nina Gierasimczuk

Turing Test



Since Turing's conceptualisations
a lot of energy in technology is put to engineering intelligent designs
which would match/outperform/mimic/complement
the behaviour of non-artificial agents (most importantly, humans).

Turing Test



Since Turing's conceptualisations

a lot of energy in technology is put to engineering intelligent designs

which would match/outperform/mimic/complement

the behaviour of non-artificial agents (most importantly, humans).

Since intelligence is displayed in problem solving the field got fragmented into different subdomains with flagship/benchmark problems for AI to defeat.

IBM's Deep Blue

- World Chess Champion (won agains Gary Kasparov in 1997).
- MINIMAX, α-β pruning, fine-tuned evaluation function and a huge library of openings.
- 500 specially designed chess processors calculating 200.000.000 moves per second.

DeepMind's AlphaGo (2016)

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
CONSERVATION
SONGBIRDS À LA CARTE
Illegal harvest of millions of Mediterranean birds
PAGE 452
RESEARCH ETHICS
SAFEGUARD TRANSPARENCY
Don't let openness backfire on individuals
PAGE 459
POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
Dawkins's calling card 40 years on
PAGE 462
NATUREASIA.COM
28 January 2016
Vol. 529, No. 7587

AlphaGo 2016 Technology

AlphaGo is a combination of machine learning and tree search:

- Monte Carlo tree search, guided by:
- a ‘value network’ and a ‘policy network’
- implemented using deep neural network technology.

AlphaGo 2016 Technology

AlphaGo is a combination of machine learning and tree search:

- Monte Carlo tree search, guided by:
- a ‘value network’ and a ‘policy network’
- implemented using deep neural network technology.

Additionally:

- Some game-specific feature detection before it’s sent to NN (*nakade*).
- Initially trained to mimic human play by (expert players from historical games), using a database of around 30 million moves.
- trained further by playing large numbers of games against other instances of itself, using reinforcement learning to improve its play.

AlphaGo 2016 Technology

AlphaGo is a combination of machine learning and tree search:

- Monte Carlo tree search, guided by:
- a ‘value network’ and a ‘policy network’
- implemented using deep neural network technology.

Additionally:

- Some game-specific feature detection before it’s sent to NN (*nakade*).
- Initially trained to mimic human play by (expert players from historical games), using a database of around 30 million moves.
- trained further by playing large numbers of games against other instances of itself, using reinforcement learning to improve its play.

To avoid ‘disrespectfully’ wasting its opponent’s time,
it resigns if its assessment of win probability falls beneath a certain threshold.

Monte-Carlo Tree Search

When designing a good evaluation function is very difficult
a (sample-based) learning approach can be used:
Monte-Carlo Tree Search (MCTS).

Monte-Carlo Tree Search

When designing a good evaluation function is very difficult
a (sample-based) learning approach can be used:
Monte-Carlo Tree Search (MCTS).

Monte-Carlo Tree Search is MINIMAX meets **reinforcement learning**:

The algorithm builds a partial game tree like Minimax,

states are evaluated by reinforcement learning.

Monte-Carlo Tree Search

When designing a good evaluation function is very difficult
a (sample-based) learning approach can be used:
Monte-Carlo Tree Search (MCTS).

Monte-Carlo Tree Search is Minimax meets **reinforcement learning**:
The algorithm builds a partial game tree like Minimax,
states are evaluated by reinforcement learning.

Differences to Minimax with cutoff:

- Non-uniform expansion of tree:
  most promising states expanded first.
- Evaluation is not hard-coded:
  based on Monte-Carlo sampling (random play-outs/simulations).

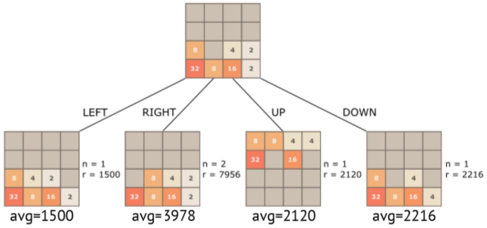
State evaluation

Evaluation of a state *s* (like in the EVAL(*s*) in Minimax with cutoff):

Play *n* **random** full games beginning in *s*, and take the average utility of those games.

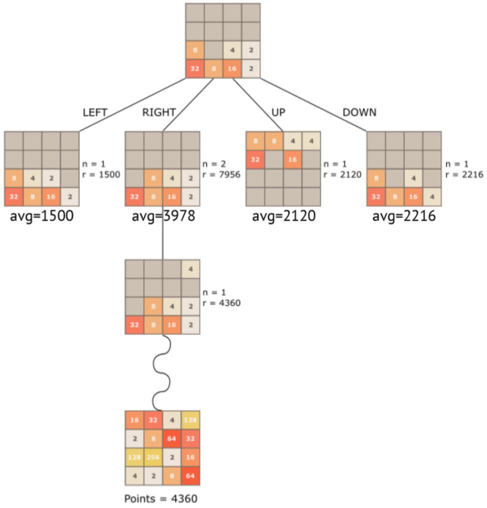
The utility is called a **reward** in Monte-Carlo Tree Search.

Example: State evaluation in 2048



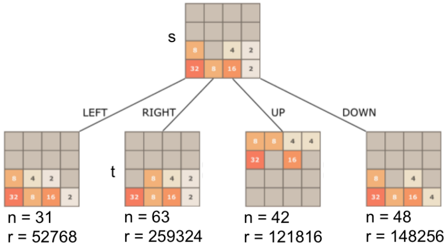
- n: number of games played through the state.
- r: Sum of rewards (scores) received in those n games.

Example: State evaluation in 2048



- *n*: number of games played through the state.
- *r*: Sum of rewards (scores) received in those *n* games.

Example: State evaluation in 2048



- n: number of games played through the state.
- r: Sum of rewards (scores) received in those n games.

Node t chosen for **expansion** is node with max Upper Confidence Bound (UCB) value:

$$UCB(t) = \frac{r_t}{n_t} + \sqrt{\frac{2 \ln n_s}{n_t}}$$

where:

- s is parent of t,
- $\frac{r_t}{n_t}$ is the current average utility (reward) of t;
- n_t and n_s are the no of times t and s were played, resp.

Monte-Carlo Tree Search

Expanding a node in Tree- and Graph-search means adding all children.

In MCTS, often *one child is added at a time*.
The **frontier** hence contains any node for which not all children have been generated.

Monte-Carlo Tree Search

Expanding a node in Tree- and Graph-search means adding all children.

In MCTS, often *one child is added at a time*.
The **frontier** hence contains any node for which not all children have been generated.

Monte-Carlo Tree Search iteratively loops through the 4 steps:

1. **Selection**:

   a node in the frontier is selected for expansion:

   start at the root and find a path to a frontier node by iteratively selecting the best child.

   use UCB to find the best child of each node on the path.

2. **Expansion**:

   expand the selected node, that is add it to the frontier.

3. **Simulation**:

   play a random game from the generated child node;

   this is called a **playout** or **rollout**.

4. **Backup**:

   update the evaluation of all nodes on the path

   from the root to the generated child node based on the playout;

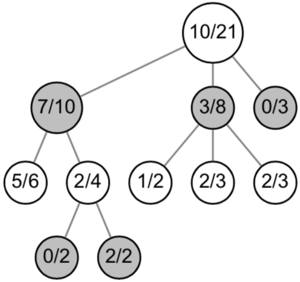
   this is called **backpropagation**.

MCTS example

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:

  x - no of white's wins through that node

  y - no of games played through that node.
- Black is minimising, as in Minimax.



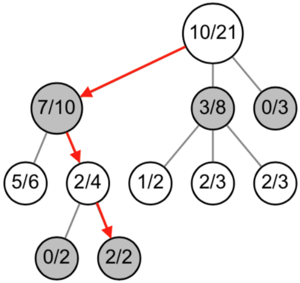
MCTS example

Step 1: Selection

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:

  x - no of white's wins through that node

  y - no of games played through that node.
- Black is minimising, as in Minimax.

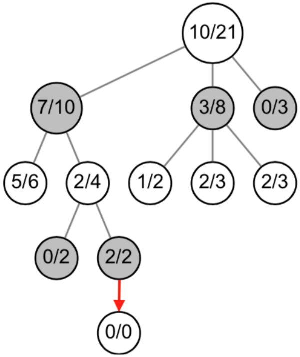


MCTS EXAMPLE
STEP 2: EXPANSION

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:

  x - no of white's wins through that node

  y - no of games played through that node.
- Black is minimising, as in Minimax.



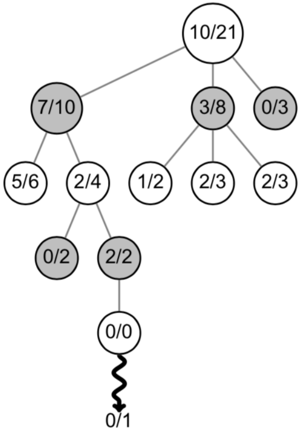
MCTS EXAMPLE

STEP 3: SIMULATION

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:

  x - no of white's wins through that node

  y - no of games played through that node.
- Black is minimising, as in Minimax.



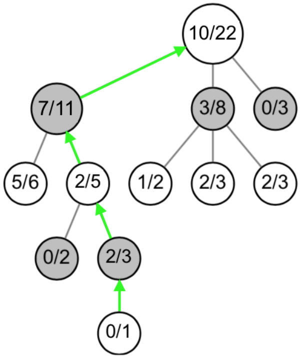
MCTS example

Step 4: Back-propagation

- Two-player game: white and black.
- White moves in the white nodes.
- Each node is labelled by x/y:

  x - no of white's wins through that node

  y - no of games played through that node.
- Black is minimising, as in Minimax.



Monte-Carlo Tree Search vs Minimax

Monte-Carlo Tree Search converges to Minimax.

Monte-Carlo Tree Search is essentially just a more advanced way to prune the search tree than Minimax with cutoff.

Monte-Carlo Tree Search vs Minimax

Monte-Carlo Tree Search converges to Minimax.

Monte-Carlo Tree Search is essentially just a more advanced
way to prune the search tree than Minimax with cutoff.

For the game of Chess
MCTS is today far superior to Minimax with cut-offs and alpha-beta pruning.

For the game of 2048
Minimax with a good evaluation function performed better than MCTS.

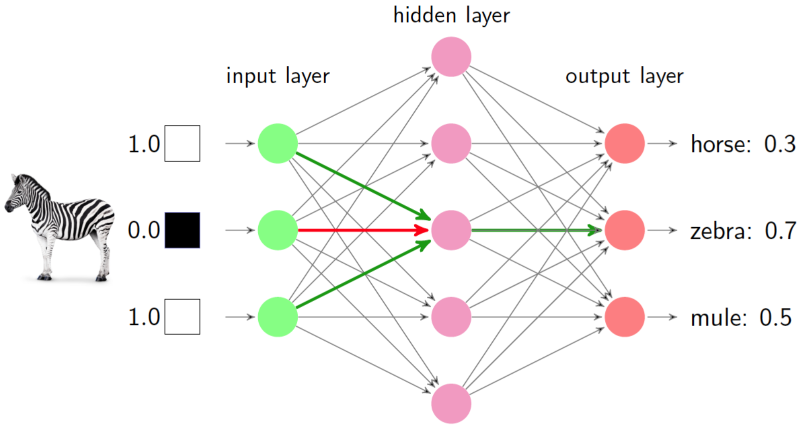
Nature video: The Computer that Mastered Go

Short Excerpt

Full Movie

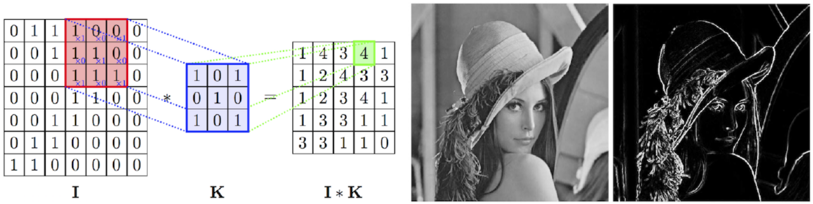
AlphaGo - The Movie

Neural Networks

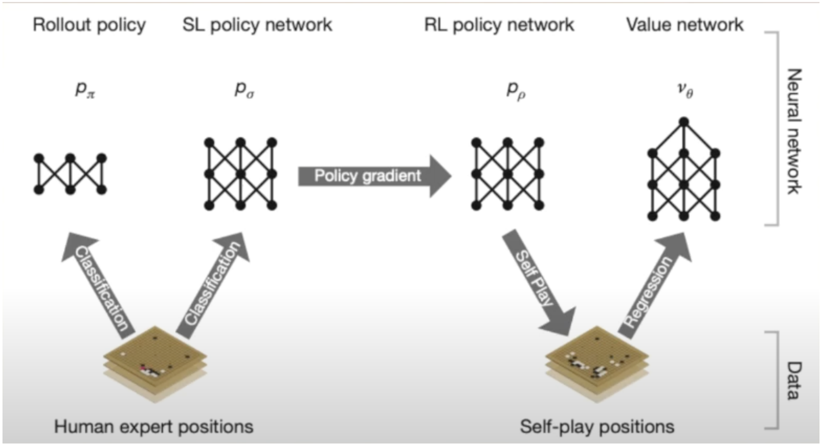


Deep Neural Networks

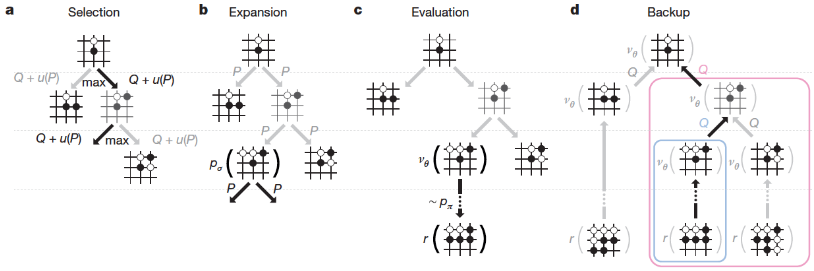
- Deep networks: many layers.
- Some layers are convolutional: sliding windows applying image filters to find geometrical features in different parts of the picture.



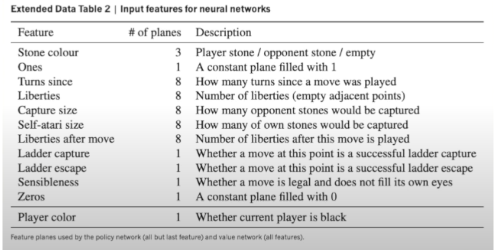
Neural Networks in AlphaGo



MCTS AlphaGo



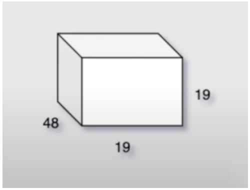
States for Policy and Value Networks



Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Feature planes used by the policy network (all but last feature) and value network (all features).



From Alpha Go to General Game Playing

General game playing (GGP)

is the design of artificial intelligence programs

to be able to play more than one game successfully

GGP is a necessary milestone on the way to Artificial General Intelligence.

From Alpha Go to General Game Playing

General game playing (GGP)

is the design of artificial intelligence programs

to be able to play more than one game successfully

GGP is a necessary milestone on the way to Artificial General Intelligence.

- AlphaGo (2016)
- AlphaGo Zero (2017) - a version without human data.
- AlphaZero (2017) - can play chess as well as Go.
- MuZero (2019) - involves model-free reinforcement learning; can handle complex inputs (visual video games).

THE END OF LECTURE 6